Extreme Extraction: Only One Hour per Relation

Raphael Hoffmann
Department of Computer Science & Engineering
University of Washington
Seattle, WA, U.S.A.
raphaelh@uw.edu

Luke Zettlemoyer
Department of Computer Science & Engineering
University of Washington
Seattle, WA, U.S.A.
lsz@cs.washington.edu

Daniel S. Weld
Department of Computer Science & Engineering
University of Washington
Seattle, WA, U.S.A.
weld@cs.washington.edu

ABSTRACT

Information Extraction (IE) aims to automatically generate a large knowledge base from natural language text, but progress remains slow. Supervised learning requires copious human annotation, while unsupervised and weakly supervised approaches do not deliver competitive accuracy. As a result, most fielded applications of IE, as well as the leading TAC-KBP systems, rely on significant amounts of manual engineering. Even "Extreme" methods, such as those reported in Freedman et al. [11], require about 10 hours of expert labor per relation.

This paper shows how to reduce that effort by an order of magnitude. We present a novel system, INSTAREAD, that streamlines authoring with an ensemble of methods: 1) encoding extraction rules in an expressive and compositional representation, 2) guiding the user to promising rules based on corpus statistics and mined resources, and 3) introducing a new interactive development cycle that provides immediate feedback — even on large datasets. Experiments show that experts can create quality extractors in under an hour and even NLP novices can author good extractors. These extractors equal or outperform ones obtained by comparably supervised and state-of-the-art distantly supervised approaches.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications - Data Mining; H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing - Linguistic Processing; I.2.7 [**Artificial Intelligence**]: Natural Language Processing - Text Analysis; I.5.5 [**Pattern Recognition**]: Implementation - Interactive Systems

General Terms

Experimentation, Human Factors

Keywords

information extraction, rule-based extraction, natural language processing, interactive systems

1. INTRODUCTION

Information Extraction (IE), the process of distilling semantic relations from natural language text, continues to gain attention. If applied to the Web, such IE systems have the potential to create a large-scale knowledge base which would benefit important tasks such as question answering and summarization.

Applying information extraction to many relations, however, remains a challenge. One popular approach is supervised learning of relation-specific extractors, but these methods are limited by the availability of training data and are thus not scalable. Unsupervised and weakly supervised methods have been proposed, but are not sufficiently accurate. Many successful applications of IE therefore continue to rely on significant amounts of manual engineering. For example, the best performing systems of the TAC-KBP slot filling challenge make central use of manually created rules [21, 36].

In response, Freedman et al. [11] proposed *Extreme Extraction*, a combination of techniques which enabled experts to develop five slot-filling extractors in 50 hours, starting with just 20 examples per slot type. These extractors outperformed ones learned with manual supervision and also required less effort, when data labeling costs were included.

In this work, we seek to dramatically *streamline* the process of extractor engineering, while handling the more general task of binary relations, $r(a, b)$, where both arguments are free. Our goal is to enable researchers to create a high-quality relation extractor in under one hour, using no prelabeled data. To achieve this goal, we propose an extractor development tool, INSTAREAD, which defines a user-system interaction based on three key properties. First, experts can write *compositional* rules in an expressive logical notation. Second, the system guides the expert to promising rules, for example through a bootstrap rule induction algorithm which leverages the distribution of the data. Third, rules can be tested instantly even on relatively large datasets.

This paper makes the following contributions:

- We present INSTAREAD, an integrated ensemble of methods for rapid extractor construction.
- We show how these components can be implemented to enable *real-time* interactivity over millions of documents.
- We evaluate INSTAREAD empirically, showing 1) an expert user can quickly create high precision rules with large recall. [1] that greatly outperform comparably supervised and state-of-the-art distantly supervised approaches and require one tenth the manual effort of Freedman's [11] approach. We also present 2) the cumulative gains due to different INSTAREAD features,

[1] All rule sets developed as part of this work, the training data produced by odesk workers, and the output extractions from each system are available upon request.

as well as 3) an error analysis indicating that more than half of extractor mistakes stem from problems during preprocessing (e.g., parsing or NER). We further show that 4) even NLP novices can use INSTAREAD to create quality extractors for many relations.

2. PROBLEM DEFINITION

Engineering competitive information extractors often involves the development of a carefully selected set of rules. The rules are then used in a number of ways, for example, to create deterministic rule-based extractors [21, 36], or as features or constraints in learning-based systems [28, 13]. Typically, however, the development of rules is an iterative process of refinement that involves (1) analyzing a development corpus of text for variations of relation mentions, (2) creating hypotheses for how these can be generalized, (3) formulating these hypotheses in a rule language, and (4) testing the rules on the corpus.

Unfortunately, each of the steps in this cycle can be very time intensive. For example, when analyzing a corpus an expert may spend much time searching for relevant sentences. When creating hypotheses, an expert may not foresee possible over- or under-generalizations. An expert's intended generalization may also not be directly expressible in a rule language, and testing may be computationally intensive in which case the expert is unable to obtain immediate feedback.

Our goal in this work is to develop and compare techniques which accelerate this cycle, so that engineering a competitive extractor requires less than an hour of expert time.

3. PRIOR WORK

Freedman et al. [11]'s landmark work on 'Extreme Extraction' first articulated the important challenge of investigating the development of information extractors within a limited amount of time. Their methods, which allowed an expert to create a question answering system for a new domain in one week, used orders of magnitude less human engineering than the norm for ACE[2], TAC-KBP and MUC[3] competitions. Key to this improvement was a hybrid approach that combined a bootstrap learning algorithm and manual rule writing. Freedman et al. showed that this combination yields higher recall and F1 compared to approaches that used only bootstrap learning or manual rule writing, but not both.

Freedman et al.'s task is related but different from our task; some of these differences make it harder and others easier. In particular, Freedman et al. sought to extract relations $R_i(arg_1, arg_2)$, in which one of the arguments was fixed. A small amount of training data was assumed for each relation, and the task included adaptation of a named entity recognizer and coreference resolution system.

With INSTAREAD we propose a combination of different, complementary techniques to those of Freedman et al., that focus on streamlining rule authoring. One of these techniques leverages a refined and very effective bootstrap algorithm that keeps the user in the loop, whereas Feedman et al.'s bootstrap learner ran autonomously without user interaction.

A large amount of other work has looked at bootstrapping extractors from a set of seed examples. Carlson et al.'s NELL [3] performs coupled semi-supervised learning to extract a broad set of instances, relations, and classes from a Web-scale corpus. Two of the four relations we report results for in Section 8 are covered by NELL, yielding 174 (attendedSchool) and 977 (married) instances after 772 iterations. To avoid a decline in precision (on average 57% after 66 iterations), NELL relies on periodic human supervision of 5 minutes per relation every 10 iterations. Other systems leverage large knowledge bases for supervision. PROSPERA [23] uses MaxSat-based constraint reasoning and improves on the results, but requires that relation arguments be contained in the YAGO knowledge base [35]. Only one of our four relations is extracted by PROSPERA (attendedSchool), yielding 1,371 instances at 78% precision. 26,280 instances of this relation from YAGO were used for supervision. DeepDive [26, 27, 30] scales a Markov logic program to the same corpus and uses Freebase for supervision. It reaches an F1 score of 31% on the TAC-KBP relation extraction benchmark. MIML-RE [37] and MultiR [14], also apply distant supervision from a knowledge base but add global constraints to relax the assumption that every matching sentence expresses a fact in the knowledge base. Except for the latter two systems, which we compared to, none of the above systems is publicly released, making a direct comparison impossible. In general, all of the above approaches suffer from relatively low recall and combat semantic drift by relying on redundancy, global constraints, large knowledge bases, or validative feedback.

While above approaches attempt to avoid manual input altogether, other approaches try to make manual input more effective. These include compositional pattern languages for specifying rule-based extractors, such as CPSL [1] and TokensRegex [4]. Rule-based extraction has also been scaled to larger datasets by applying query optimization [17, 6]. Unlike our work, this line of research does not evaluate the effectiveness of these languages with users, in terms of development time and extraction quality. Another approach to human input is active learning. Miller et al. [20] learn a Perceptron model for named-entity extraction; unlabeled examples are ranked by difference in perceptron score. Riloff [32] proposes an approach to named-entity extraction, which requires users to first classify documents by domain, and then generates and ranks candidate extraction patterns. Active learning has also been studied in more general contexts, for learning probabilistic models with labeled instances [38] or labeled features [9]. This work differs from approaches based on active learning in at least two ways. First, they have not been evaluated on relation extraction tasks. Second, and more importantly, their general approach is to consider a particular type of feedback and then develop algorithms for learning more accurately from such feedback. In contrast, our approach is not to compare algorithms, but to compare different types of manual feedback.

4. OVERVIEW OF INSTAREAD

To evaluate techniques for accelerating the development process of rule-based extractors, we developed INSTAREAD, an interactive extractor development tool. INSTAREAD is designed to address the inefficiencies identified in Section 2 with the combination of an expressive rule language, data-driven guidance to new rules, and instant rule execution.

We next present an example of an expert interacting with the system, and then in the following sections show how INSTAREAD enables its three key properties.

4.1 Example

Anna wants to create an extractor for the killed(killer,victim) relation. After selecting a development text corpus she proceeds as follows.

- To find example sentences, Anna searches for sentences containing keyword 'killed' (Figure 1 a). INSTAREAD suggests

[2] http://www.itl.nist.gov/iad/mig/tests/ace/

[3] http://www-nlpir.nist.gov/related_projects/muc

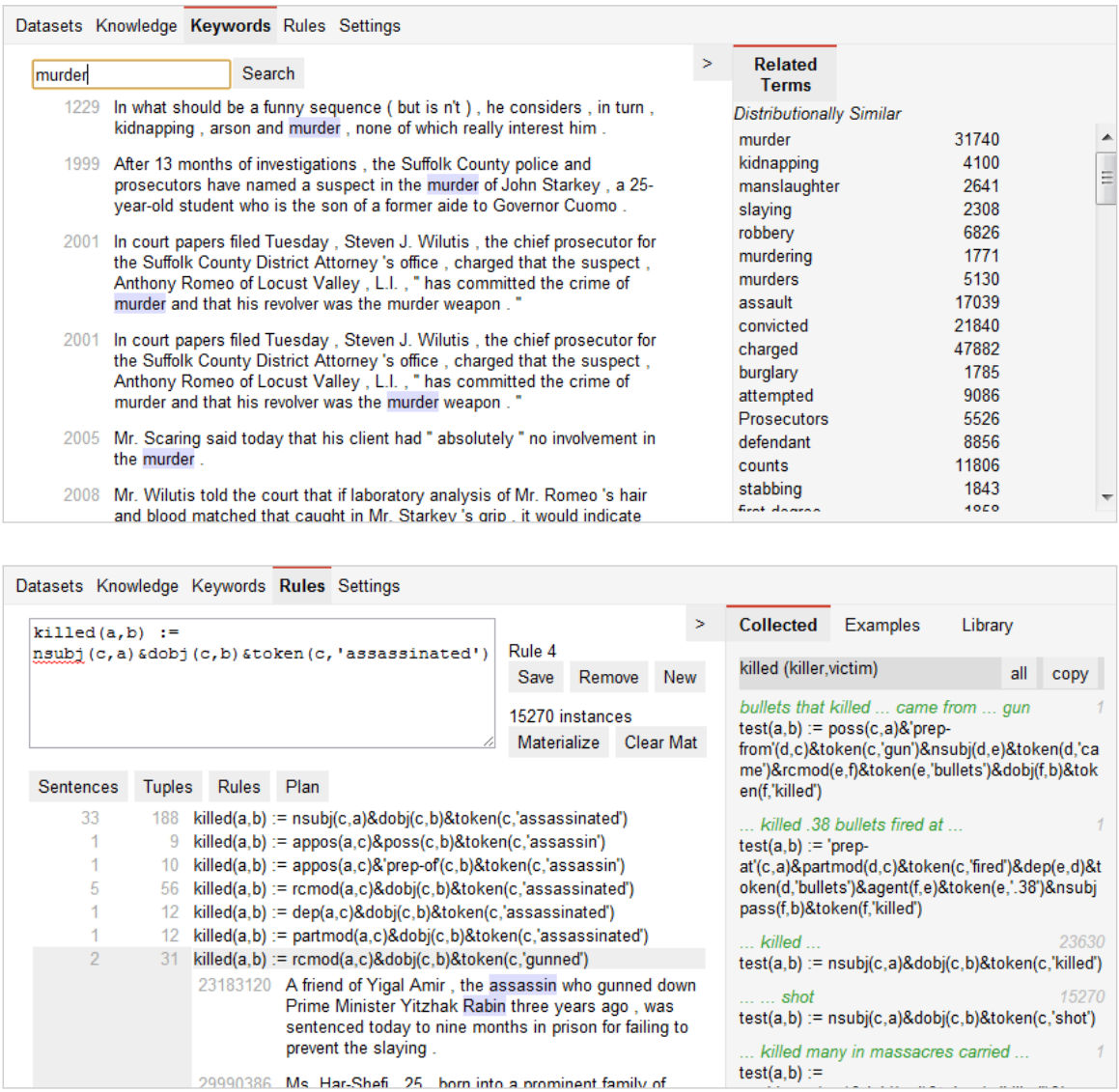


Figure 1: Selected interactions (Section 4). INSTAREAD **guides users to sentences and rules by distributionally-similar words (top), and bootstrap pattern learning (bottom).**

to also consider distributionally similar keywords such as 'murder' or 'assassin'. Within seconds Anna obtains many relevant examples.

- Anna compares examples, investigating their syntactic structure obtained by a parser, and encodes an extraction pattern as a rule: $\mathsf{killed}(a,b) \Leftarrow \mathsf{nsubj}(c,a)\&\mathsf{dobj}(c,b)\&\mathsf{token}(c,'\mathsf{murdered}')$. The system offers to automatically generalize that rule, so that it also covers the passive form as well as all tenses.
- Anna now has a working extractor which she would like to refine. INSTAREAD's bootstrapping method presents her a ranked list of new candidate rules based on the extractions of her existing rule set. Anna inspects matches of the suggested rules and selects several (Figure 1 b).
- Looking at the rules collected so far, Anna notices that many are similar, differing only in the verb that was used. She decides to refactor her rule set, so that one rule first identifies relevant verbs, and others syntactic structure. Her rule set is now more compact and generalizes better.

5. CONDITION-ACTION RULES IN LOGIC

A rule language should be both simple and expressive, so that the interaction with the system is quick and direct. To fulfill this requirement, INSTAREAD uses condition-action rules expressed in first-order logic, combined with a broad and expandable set of built-in logical predicates. For tractability, INSTAREAD requires that rules translate into *safe* domain-relational calculus [39].

Although such rules could be used to generate statistical models [8], we currently assume that all rules are deterministic and are executed in a defined order, and leave the integration with learning-based techniques as future work.

Figure 2 presents an example rule set and how it is applied to a sentence. The predicates used in this example have arguments

that range over tokens and token positions. Rules are used to define predicate killOfVictim and that predicate then gets re-used in other rules to define predicate killed. We call this ability INSTAREAD's *Composition* feature.

To increase the expressiveness of the language, INSTAREAD implements around one hundred built-in predicates, such as tokenBefore and isCapitalized. In addition, it makes available predicates that encode the output of (currently) four NLP systems, including a phrase structure parser [5], a typed dependency extractor [19][4], a coreference resolution system [29], and a named-entity tagger [10]. This allows users to write rules which simultaneously use parse, coreference, and entity type information.

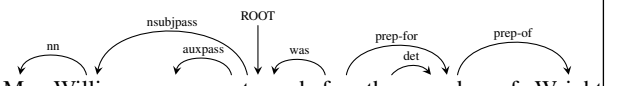
(a) Rules. Dependency predicates nn, poss, nsubjpass, prep-by, prep-for, prep-of and named-entity predicate person are precomputed.

killNoun('murder')
killNoun('assassination')
killNoun('killing')
killNoun('slaughter')

killOfVictim(c, b) ⇐ prep-of(c, b)∧token(c, d)∧killNoun(d)
killOfVictim(c, b) ⇐ nn(c, b) ∧ token(c, d) ∧ killNoun(d)
killOfVictim(c, b) ⇐ poss(c, b) ∧ token(c, d) ∧ killNoun(d)

killed(a, b) ⇐ person(a) ∧ person(b) ∧ nsubjpass(c, a)∧
token(c, 'sentenced') ∧ prep-for(c, d)∧
killOfVictim(d, b)
killed(a, b) ⇐ person(a) ∧ person(b) ∧ prep-by(c, a)∧
killOfVictim(c, b)

(b) Example sentence with typed dependencies.



(c) Predicted ground instances. Here, murder-7 refers to the 7th token in the example sentence.

killOfVictim(murder-7, Wright-10)
killed(Williams-2, Wright-10)

Figure 2: Example rule set executed on sentence.

The rules in Figure 2 are all Horn clauses, but INSTAREAD also supports disjunction (∨), negation (¬), and existential (∃) and universial (∀) quantification. While one does not often need these operators, they are sometimes convenient for specific lexical ambiguities. For example, in our evaluation discussed in section 8, one user of INSTAREAD created the following rule to extract instances of the founded(person,organization) relation:

founded(a, b) ⇐nsubj(c, a) ∧ dobj(c, b) ∧ token(c, 'built')∧
person(a) ∧ organization(b)

This rule was designed to match sentences such as: 'Michael Dell built his first company in a dorm-room.' However, this rule also incorrectly matches a number of other sentences such as: 'Mr. Harris built Dell into a formidable competitor to IBM.' While 'building an organization' typically implies a founded relation, 'building an organization *into* something' does not. This distinction can be captured in our rule by adding the conjunct ¬(∃d : prep-into(c, d)).

[4]INSTAREAD uses collapsed dependencies with propagation of conjunct dependencies.

6. GUIDING EXPERTS TO EFFECTIVE RULES

While the rule language is important for developing extractors, many hours of testing on early prototypes of INSTAREAD showed that it is not sufficient for an effective interaction. With a growing number of rules, users find it increasingly difficult to identify rules for refinement. More importantly, users don't know where to focus their attention when trying to find effective rules to add.

Feedback.

Our small example in Figure 2 already demonstrates the problem: What exactly does each rule do? How much data does each rule affect? In early testing, we noticed that users would often write short comments for each rule, consisting of the surface tokens matched. We therefore designed a technique to automatically generate such comments (depicted in Figure 3) by retrieving matched sentences, identifying sentence tokens that were explicitly referenced by one of the predicates, and concatenating the tokens in the order that they appear in a sentence. Included are '…' placeholders for the arguments of the rule's target predicate. Figure 3 also shows how INSTAREAD displays the number of matches together with each rule, eg. 257 for '…stabbed …', helping users quickly judge the importance of a rule. Although one may also be interested in precision, that cannot be obtained without annotated data. INSTAREAD also includes visualizations for dependency trees, parse trees, and coreference clusters. Such visualizations do not always convey all information encoded in the logical representation, but convey (approximate) meaning or relevance quickly.

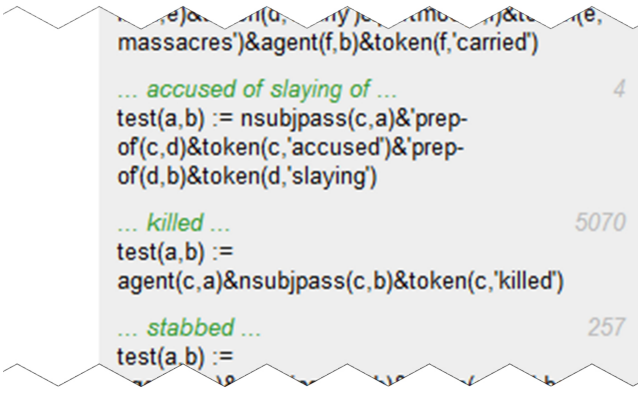


Figure 3: INSTAREAD shows automatically generated comments and number of extractions together with each rule, allowing users to see (approximate) meaning and relevance without needing to read logical expressions.

Bootstrap Rule Induction.

How does an expert know what rules to write? Coming up with good candidates is surprisingly difficult. One approach is automatic rule suggestions based on statistics. This can be done, for example, using a semi-supervised bootstrap pattern learning algo-

rithm. Freedman et al. [11] applied such an algorithm, too, but found that it was not competitive with manual pattern writing, especially with regards to recall. INSTAREAD's bootstrap algorithm therefore makes several changes: First, it instantly returns ranked bootstrap results over a large corpus. Second, it takes into account coreference information to expand recall (similarly to Gabbard et al. [12]). Third, it puts the user into the loop, allowing her to select appropriate rules after each iteration.

In particular, INSTAREAD's bootstrap technique takes as input a binary relation predicate $r(a, b)$ together with a set of rules R defining instances of r. The output is a ranked list of candidate rules S. The algorithm works by first identifying mentions of r using the existing rules R and generating the pairs (a_s, b_s) of argument surface strings of these mentions. This set of pairs is then matched to the entire corpus, retrieving all sentences containing both strings. Similar to DIRT [18], INSTAREAD then generates syntactic-lexical extraction patterns from these matches. Loosely following Mintz et al. [22], the system finds a path of syntactic dependencies connecting the matched surface strings, and then creates a rule that is a conjunction of syntactic dependencies and lexical constraints on that path, as well as entity type constraints (if activated by user). For examples, refer to Figure 1.

Rule suggestions, S, are sorted by two scores: pointwise mutual information of the suggested rule with the original rule set R, and number of extractions of suggested rule. The latter may show more irrelevant rules on top, but the relevant ones among them have many extractions often reducing overall effort. Users can switch between the sort orders.

Word-level Distributional Similarity.

Although our enhancements to the bootstrap approach may increase recall, recall is still limited since bootstrap requires that the same tuples appear multiple times in the corpus. To help experts find additional relation mentions, INSTAREAD therefore also includes another shallow technique: keyword search combined with keyword suggestions.

Keywords are suggested based on distributional similarity to a seed keyword. For example, the seed 'murdered' returns 'assassinated', 'slayed', 'shot', and more. Specifically, each word w in the text corpus is represented as a vector of weighted words v co-occurring in sentences with w. The similarity of two words w_1, w_2 is then defined as the cosine similarity of their vector representations. An additional list of keyword suggestions shows keywords which contain the seed keyword as prefix. Suggested keywords are always displayed together with their number of occurrences in the corpus to guide users to the most relevant keywords.

Although this keyword-based approach may be effective in finding relevant sentences, early experiments have shown that a long time is spent to writing extraction rules based on those sentences. We therefore added a simple interface feature: Experts could click on words to indicate relation arguments, and the system will generate rule candidates using our bootstrap generation algorithm.

Core Linguistic Rules.

The final problem we address is the fact that a relation extractor typically needs a large number of rules that are not specific to the relation. For example, there exist many syntactic variations that follow common linguistic patterns. To reduce effort, we seek to populate the system with such general rules right from the start.

In a first step, we encoded a set of grammatical rules: Given a verb base form, INSTAREAD can generate rules encoding syntactic-lexical patterns for 182 combinations of tense, voice, and person. For example, given subject X, object Y and verb 'kill', the system generates rules to capture phrases such as 'Y was killed by X', 'X regretted killing Y', 'X would later kill Y'. To avoid inaccuracies from using a stemmer, INSTAREAD includes inflection rules and a corpus of inflections for 16851 verbs mined from Wiktionary. This grammatical background knowledge is provided to the user through a set of additional built-in predicates.

7. EFFICIENT RULE EVALUATION

To enable its interactivity, INSTAREAD must evaluate rules and guide users to effective rules very quickly, even with compositional rules and large datasets.

INSTAREAD is built on top of an RDBMS. Variables in its logical expressions are assigned a data type that can be Pos (token position), Span (token span), Int (integer), Str (string), or Ref (reference). Each of these data types is internally mapped to a composite SQL data type. For example, token spans are mapped to the SQL types integer, byte, byte, where the first is used to identify a sentence and the others indicate start and end positions within a sentence. Predicates are either *extensional* or *intensional*. Extensional ones materialize instances in relational tables, while intensional ones are defined by (partial) SQL queries. An example of an extensional predicate is our killed(a, b) extractor, which stores the result set of the extraction rules depicted in Figure 2. An example of an intensional predicate is str2span(s, t) which returns all mentions of a multi-word string using an inverted index. For details on how this predicate gets translated into SQL, see Figure 5 in the appendix. The key component of INSTAREAD's implementation is its translation of logical rules into SQL queries. The system first parses logical rules into an abstract syntax tree (AST). To ensure that the rules do not yield infinite result sets and can be translated into SQL, it checks for *safety* [39]. It then infers variable types and links predicates, then translates into an AST of tuple relational calculus, and eventually SQL, following the algorithms described in [39]. For an example translation, see Appendix A.

For performance, INSTAREAD creates a BTree index for each column of an extensional predicate. Built-in predicates (which tend to contain more instances, e.g. all syntactic dependencies), also use multi-column indices. A variety of information is pre-computed on a Hadoop cluster, including phrase structure trees, dependencies, coreference clusters, named-entities, rule candidates for bootstrapping, and distributionally similar words.

This large number of indices and pre-computed information is important because INSTAREAD does not constrain the set of queries and most queries touch the entire text corpus. It also allows each iteration of bootstrapping to be performed by a single SQL query. Across all of the experiments the median query execution time was 74ms. Achieving such interactivity is crucial for quickly building accurate extractors.

8. EXPERIMENTS

In our evaluation, we measured if INSTAREAD's features enable an expert to create quality extractors in less than one hour, and which of the features contribute most to reducing effort. We also report on an error analysis to get insights into potential future improvements. Finally, we report early results of a follow-up experiment, in which we evaluated INSTAREAD's usability among engineers without NLP background.

8.1 Experimental Setup

We evaluated the performance of an expert in a controlled experiment, in which the expert user was given one hour of time per relation to develop four relation extractors. Besides descriptions of

the four relations and a corpus of (unlabeled) news articles, which was loaded into INSTAREAD, no other resources were provided. Our expert was familiar with INSTAREAD and NLP in general, but had no experience with the relations tested. All user and system actions were logged together with their timestamps.

We were interested in determining the effectiveness of four of INSTAREAD's features: bootstrap rule induction *(Bootstrap)*, word-level distributional similarity *(WordSim)*, core linguistic rules *(Linguistics)*, and the power of rule (de-)composition *(Composition)*. In order to more easily measure the impact of each of these features, our user was required to use only one at a time and switch to the next at given time intervals (Figure 4).

Baselines.

We compare the performance of the extractors created with our proposed system INSTAREAD to three baselines.

MIML-RE [37] and **MultiR** [14] are two state-of-the-art systems for learning relation extractors by distant supervision from a database. As a database we use the instances of the relations contained in Freebase [2]. Negative examples are generated from random pairs of entity mentions in a sentence.[5]

SUP is a supervised system which learns a log-linear model using the set of features for relation extraction proposed by Mintz et al. [22]. The supervision is provided by four annotators hired on `odesk.com` who rated themselves as experts for data entry, and were encouraged to use any tool of their choice for annotation. Each annotator was asked to spend 1 hour per relation to identify sentences in the development corpus containing that relation and marking its arguments. To control the variation related to the order in which relations were presented (users get faster with time), we used a Latin square design and paid for 1 additional hour before the experiment to allow users to get familiar with the task. Negative examples were added as in the distantly supervised cases.

Datasets.

We used the New York Times Annotated Corpus [34] comprising 1.8M news articles (45M sentences) published between 1987 and 2007. A random half of the articles were used for development, the other half for testing.

Relations.

We selected four relations: attendedSchool (person,school), founded (founder,organization), killed (killer,victim), and married (spouse1,spouse2). These relations were selected because they cover a range of domains, they were part in previous evaluations [16, 7, 33], and they do not require recognition of uncommon entity types.[6] For preprocessing, we used the CJ Parser [5] and Stanford's dependency [19], coreference [29], and NER [10] systems.

[5]We added negative examples at a ratio of 50:1 to positives. Increasing this ratio increases precision but reduces the number of extractions, while decreasing has the opposite effect. We found that this setting provided a better trade-off than the default used by these distantly supervised systems on the data by Riedel et al. [31], which returned no extractions in our case.

[6]INSTAREAD's bootstrap rule induction and core linguistic rules currently only target binary relations, but not entity types. To identify named entities of types person and organization, we thus used the Stanford NER system. To handle relation attendedSchool we additionally created a recognizer for type school by listing 30 common head words such as 'University' before the experiment. This process took under 5 minutes.

	attendedSchool	founded	killed	married
InstaRead (rules)	94	97	141	48
SUP (examples)	68	79	36	52

Table 2: Manual input generated in one hour of time. In the supervised case, annotators had difficulty finding examples for the *killed* relation which had fewer mentions in the corpus. In contrast, INSTAREAD's effort-reducing features, such as rule suggestions, made it easy to find examples and add relevant rules quickly. Our user of INSTAREAD actually generated more rules for this relation, in the allotted time, due to the larger number of syntactic variations.

Metrics.

Extractions were counted on a mention level, which means that an extraction consisted of both a pair of strings representing named entities as well as a reference to the sentence expressing the relation. To measure precision, we sampled 100 extractions and manually created annotations following the ACE guidelines [7].

8.2 Comparing InstaRead to Baselines

Overall results are summarized in Table 1. In the case of INSTAREAD, precision was 90% or higher[7] for each of the four relations, and each extractor returned thousands of tuples. For the three baselines, results varied between relations but in all cases significantly fewer extractions were returned, and in all but two cases precision was significantly lower. The most challenging of the four relations was *killed*, since it can be expressed in many different ways, and many such expressions have multiple meanings. At the same time, mentions of the *killed* relation occur less frequently in the corpus than mentions of the other three relations. The supervised baseline did not return results, and the distantly supervised systems could not be applied because Freebase did not contain instances for the *killed* relation.

User Feedback.

Looking more carefully at the feedback supplied by our users, we found that one hour use of INSTAREAD yielded 95 rules on average. This compares to an average of 59 examples per hour annotated by users in the supervised case. INSTAREAD's effort-reducing features made it easy to find relevant sentences and add rules quickly, which frequently only required confirmation of a system-generated suggestion. Users in the supervised case had difficulty finding sentences expressing the relations. Two of the annotators reported that they started off reading the text corpus linearly, but barely found any examples that way. They later searched by keywords ('College') and wildcards ('marr*'). With 77 and 89 examples per hour these users found more examples (but not necessarily more variations) than users who scanned the corpus linearly

[7]The high precision in the case of INSTAREAD may seem surprising, but is in fact easy to attain for many relations. Since every change the user makes to the rule set immediately triggers a re-evaluation and visual presentation of extractions and their sentences, the user can quickly adapt the rule set until she is satisfied with precision on the training set. There is generally little overfitting, due to the training set being large and the rules not being automatically selected but created by a human with intuitions about language.
This contrasts with SUP, where a fixed feature set leads to high precision on one relation (founded), but low precision on another (attendedSchool). Without interactive feedback, it is very challenging to create an effective feature set as well as create effective annotated examples, especially negative ones.

	attendedSchool		founded		killed		married	
	Pr	#e	Pr	#e	Pr	#e	Pr	#e
InstaRead	**100%**	**52,338**	91%	**20,733**	**90%**	**4,728**	90%	**63,742**
MIML-RE	9%	14,960	28%	14,960	N/A	0*	**93%**	9,900
MultiR	26%	18,480	38%	10,340	N/A	0*	51%	24,200
SUP	12%	25,196	**100%**	2,255	N/A	0	44%	7,867

Table 1: Precision (Pr) and number of extractions (#e) for the NYTimes test dataset. *Cases where extraction could not be performed because no target database could be found that contained examples required for distant supervision.

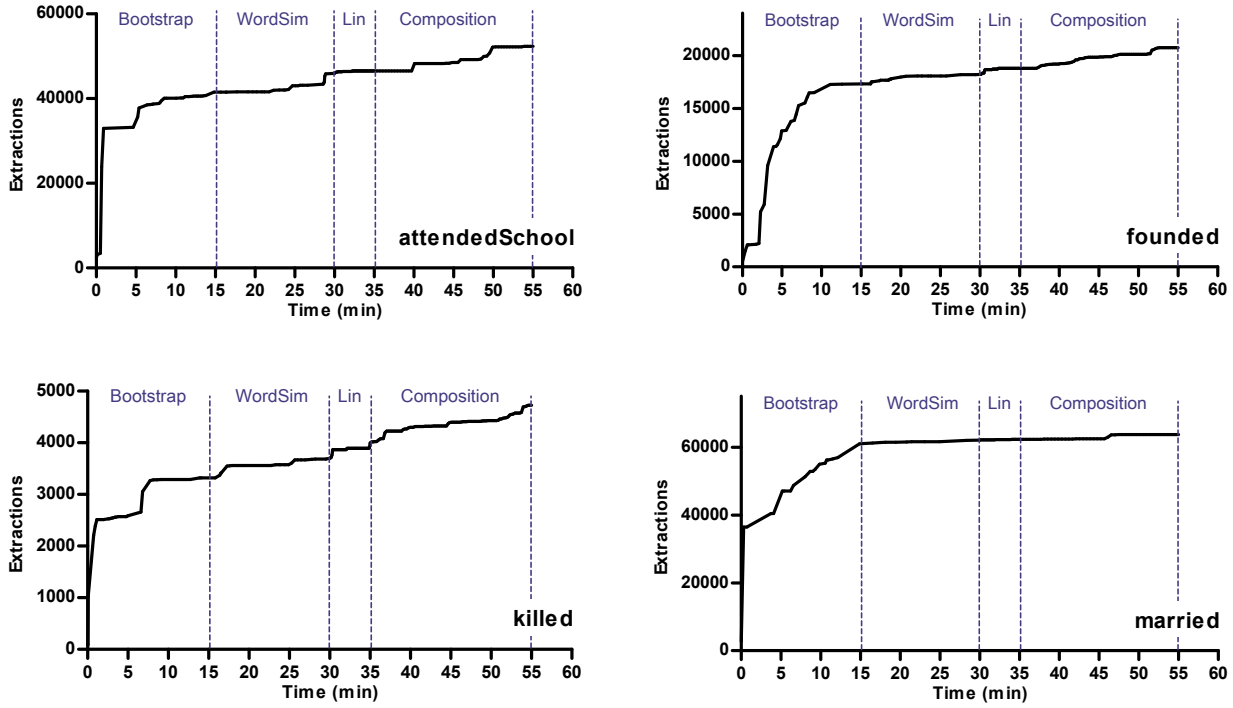


Figure 4: Number of extractions on an independent test set while using INSTAREAD for 55 minutes. *Bootstrap* (Section 6) captures a large number of extractions quickly, but does not yield additional gains after a few minutes. *WordSim* (Section 6) enables slow, but consistent gains. *Linguistics* (Section 6) provides a small gain. *Composition* (Section 5) is helpful when there exist a large number of lexical stems that imply a relation (e.g. for the killed relation).

and found 19 and 51 examples per hour.

Table 2 shows a breakdown by relation, and reveals a striking difference between INSTAREAD and SUP for the *killed* relation. In the supervised case, users were able to identify far fewer examples for this relation than others. In contrast, our user of INSTAREAD actually generated most rules for this relation. This shows that INSTAREAD did not suffer from the problem of finding examples. In fact, as we will see, INSTAREAD's effort-reducing features were actually most effective for this relation, and the larger number of rules was necessary to cover a larger set of variations.

Impact of Effort-Reducing Features.

Figure 4 shows the contribution of each of the four features on number of extractions. The vast majority of extractions, 84%, were obtained by rules created during the *Bootstrap* phase. *Bootstrap* has the ability to aggregate over many potential rules and then rank those taking into account the number of extractions. This ranking ensures that user effort is directed to rules which are likely to matter most. Such ranking is not possible with the *WordSim* feature, which, however, has a different advantage: It can find rarely used ways of expressing a relation. In contrast, *Bootstrap* only works if the same relation instance is expressed multiple times in different ways. We therefore often observe that it provides no more improvement after a few minutes of use. 3.4% of extractions were obtained by rules created during the *WordSim* phase, 2.6% during the *Linguistics* phase, and 9.5% during the *Composition* phase.

Figure 4 further reveals differences between relations. For *married*, the relatively small number of common variations were already captured in only 15 minutes, after which *WordSim*, *Linguistics*, and *Composition* features provided little benefit. For *killed*, however, each of the four effort-reducing features substantially increased the number of extractions. This shows that INSTAREAD's ensemble of effort-reducing features was effective in guiding our user to the many variations of the *killed* relation.

Analysis of INSTAREAD*'s Errors.*

INSTAREAD's precision errors for the four relations were to a large degree caused by errors in preprocessing, especially dependency extraction (55%) and NER (24%). Only 21% of precision errors were caused by overly general rules that the expert user had

developed. All were due to ambiguities of the words fell, executor, and built. While the effort-reducing features have been designed to increase recall, INSTAREAD's focus on only deterministic rules is not adequate to easily handle such ambiguities – a shortcoming we would like to address in future work.

Enhancing Supervised Extraction.

Finally, we are interested in knowing if an increase in time would let users in the supervised case match INSTAREAD's results. We therefore combined the annotations of all four annotators; each relation's examples thus corresponded to four hours of manual effort. Trained on this data, SUP returned more extractions (attendedSchool – 51,492, founded – 7,482, killed – 220, married – 24,866), but precision remained low and in two cases even decreased slightly (attendedSchool – 12%, founded – 97%, killed – 0%, married – 34%). In summary, additional time *does* improve performance, but many more hours of annotation effort would be required to reach performance comparable to INSTAREAD.

The features we selected have been shown to work well for many relations [22], but it is still possible that better features could improve the supervised learning algorithm's performance. However, feature engineering itself takes considerable effort, usually measured in weeks, which would defeat our goal of building complete extractors quickly. It will be an important area for future work to determine if INSTAREAD can be adapted to support rapid authoring of rules that define feature templates, perhaps providing even better overall performance on a limited engineering budget.

Comparing to Extreme Extraction Work.

It is impossible to compare directly to Freedman et al [11], since we were unable to acquire their datasets. While their approach yielded an average precision of 53% across 5 relations, they used 50 hours of manual engineering and furthermore those hours were spread across several different experts, each with knowledge of a specific tool.

Unlike INSTAREAD's *Bootstrap* feature, their bootstrap learner ran autonomously without user interaction, but contributed little to increase overall performance. We suspect that INSTAREAD's user in the loop, instant execution, integration of coreference information, and larger corpus contributed to perceived differences in effectiveness. Section 3 discusses further differences and similarities of Freedman et al's work and INSTAREAD.

8.3 Real-world Use By Engineers

Our experiments so far tested INSTAREAD's effectiveness for a trained expert; in our final experiment, we evaluated if the system was also usable by engineers without NLP background.

We recruited four senior undergraduate students in Computer Science who used INSTAREAD as part of a quarterly class project to develop 30 relation extractors for the TAC-KBP slot filling challenge. In six meetings, usage of the tool was explained and qualitative feedback collected.

All four subjects were able to use the system with little instruction, all were able to develop extractors, and all four subjects reported that the tool made it easier for them than if they had to write their own code. Among the 27 extractors that were created, median precision was 94% (mean 75%), and median number of extractions on NYTimes data was 2283 (mean 8741). For two relations, no extractor was created due to the difficulty in creating custom entity type recognizers, and for one relation due to an implementation error. Mean precision was negatively affected by six relations which required custom entity type recognizers. INSTAREAD currently has no support for developing entity type recognizers, a shortcoming which we would like to address in future work. Another important area for improvement is the interface to manage sets of rules. The subjects found it was often easier for them to manage rule sets in code (as strings of logical expression), because they could add their own comments, re-arrange, and keep track of multiple versions.

9. CONCLUSIONS AND FUTURE WORK

Many successful applications of IE rely on large amounts of manual engineering, which often requires the laborious selection of rules to be used as extraction patterns or features.

This paper presents ways to streamline this process, proposing an ensemble of methods that enable three properties: an expressive rule language, guidance that leads users to promising rules, and instant rule testing. Our experiments demonstrate that INSTAREAD enables experts to develop quality relation extractors in under one hour – an order of magnitude reduction in effort from Freedman et al. [11]. To stimulate continued progress in the area, we release our data as explained in footnote 1.

The experiments also point to two promising directions to further reduce manual effort:

Richness of Interactions.

With the *Bootstrap*, *WordSim*, *Linguistics*, and *Composition* features, INSTAREAD offered a variety of interactions, all of which contributed to increased recall while maintaining high-precision. *Bootstrap* was particularly effective, but did not allow further improvements after a few minutes of use. *WordSim* did not show this problem, but expanded recall more slowly. *Composition* was very effective for some relations. *Linguistics* yielded smaller gains, but required less effort. Future improvements to cover additional syntactic variations, such as participle phrases, may increase gains.

We consider such variety of interactions essential, and thus plan to include interactions for clustering phrases, providing databases of instances for distant supervision, editing ontologies, providing validative feedback, and annotating sentences. Determining the relative importance of such interactions will be an important future challenge.

Deep Integration of Algorithms.

Perhaps even greater potential, however, may lie in more tightly integrating INSTAREAD's components. Our analysis of precision errors revealed that the majority of precision errors were caused by inaccurate preprocessing, and we believe that jointly taking into account manually created rules as well as the k best outputs of the preprocessing components could improve results. We further suspect learning-based techniques may be particularly important for tasks such as NER, where there exist many ambiguities, while rule-based techniques may work well for tasks such as defining implicature between phrases.

INSTAREAD's *Boostrap* feature could also be improved. It currently already leverages coreference clusters and syntactic dependencies. In fact, coreference information which greatly increases recall may explain much of bootstrap learning's observed high effectiveness compared to Freedman et al.'s work. In the future, we would like to enable *Bootstrap* to also take into account our core linguistic rules and the ability to decompose rules. Such integration may expand recall, and interestingly, might also simplify the interaction with the user. Since the integrated components enable rules with higher coverage, fewer, more distinct rules would be returned.

10. REFERENCES

[1] D. E. Appelt and B. Onyshkevych. The common pattern specification language. In *Proceedings of a Workshop Held at TIPSTER 98*, pages 23–30. Association for Computational Linguistics, 1998.

[2] K. D. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD Conference*, pages 1247–1250, 2008.

[3] A. Carlson, J. Betteridge, E. R. H. Jr., and T. M. Mitchell. Coupling semi-supervised learning of categories and relations. In *Proceedings of the NAACL HLT Workskop on Semi-supervised Learning for Natural Language Processing*, 2009.

[4] A. X. Chang and C. D. Manning. TokensRegex: Defining cascaded regular expressions over tokens. Technical Report CSTR 2014-02, Department of Computer Science, Stanford University, 2014.

[5] E. Charniak and M. Johnson. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, 2005.

[6] L. Chiticariu, R. Krishnamurthy, Y. Li, S. Raghavan, F. Reiss, and S. Vaithyanathan. Systemt: An algebraic approach to declarative information extraction. In *Proceedings of the Annual Meetings of the Association for Computational Linguistics (ACL)*, pages 128–137, 2010.

[7] G. Doddington, A. Mitchell, M. Przybocki, L. Ramshaw, S. Strassel, and R. Weischedel. Ace program - task definitions and performance measures. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC)*, pages 837–840, 2004.

[8] P. Domingos and D. Lowd. *Markov Logic: An Interface Layer for Artificial Intelligence*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2009.

[9] G. Druck, B. Settles, and A. McCallum. Active learning by labeling features. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 81–90, 2009.

[10] J. R. Finkel, T. Grenager, and C. Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 363–370, 2005.

[11] M. Freedman, L. A. Ramshaw, E. Boschee, R. Gabbard, G. Kratkiewicz, N. Ward, and R. M. Weischedel. Extreme extraction - machine reading in a week. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1437–1446, 2011.

[12] R. Gabbard, M. Freedman, and R. M. Weischedel. Coreference for learning to extract relations: Yes virginia, coreference matters. In *Proceedings of the Annual Meeting of the Association for Computation Linguistics (ACL)*, pages 288–293, 2011.

[13] K. Ganchev, J. Graça, J. Gillenwater, and B. Taskar. Posterior regularization for structured latent variable models. *Journal of Machine Learning Research*, 11:2001–2049, 2010.

[14] R. Hoffmann, C. Zhang, X. Ling, L. S. Zettlemoyer, and D. S. Weld. Knowledge-based weak supervision for information extraction of overlapping relations. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 541–550, 2011.

[15] R. Hoffmann, C. Zhang, and D. S. Weld. Learning 5000 relational extractors. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 286–295, 2010.

[16] H. Ji, R. Grishman, and H. T. Dang. An overview of the tac2011 knowledge base population track. In *Proceedings of the Text Analysis Conference (TAC)*, 2011.

[17] R. Krishnamurthy, Y. Li, S. Raghavan, F. Reiss, S. Vaithyanathan, and H. Zhu. Systemt: a system for declarative information extraction. *SIGMOD Record*, 37(4):7–13, 2008.

[18] D. Lin and P. Pantel. Dirt - discovery of inference rules from text. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 323–328, 2001.

[19] M.-C. D. Marneffe, B. Maccartney, and C. D. Manning. Generating typed dependency parses from phrase structure parses. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC)*, 2006.

[20] S. Miller, J. Guinness, and A. Zamanian. Name tagging with word clusters and discriminative training. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL)*, 2004.

[21] B. Min, X. Li, R. Grishman, and A. Sun. New york university 2012 system for kbp slot filling. In *Proceedings of the Text Analysis Conference (TAC)*, 2012.

[22] M. Mintz, S. Bills, R. Snow, and D. Jurafsky. Distant supervision for relation extraction without labeled data. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1003–1011, 2009.

[23] N. Nakashole, M. Theobald, and G. Weikum. Scalable knowledge harvesting with high precision and high recall. In *Proceedings of the International Conference on Web Search and Data Mining (WSDM)*, pages 227–236, 2011.

[24] N. Nakashole, G. Weikum, and F. M. Suchanek. Discovering and exploring relations on the web. *The Proceedings of the VLDB Endowment (PVLDB)*, 5(12):1982–1985, 2012.

[25] N. Nakashole, G. Weikum, and F. M. Suchanek. Discovering semantic relations from the web and organizing them with PATTY. *SIGMOD Record*, 42(2):29–34, 2013.

[26] F. Niu, C. Zhang, C. Re, and J. W. Shavlik. Deepdive: Web-scale knowledge-base construction using statistical learning and inference. In *In Proceedings of the Second International Workshop on Searching and Integrating New Web Data Sources (VLDS)*, pages 25–28, 2012.

[27] F. Niu, C. Zhang, C. Ré, and J. W. Shavlik. Elementary: Large-scale knowledge-base construction via machine learning and statistical inference. *International Journal on Semantic Web and Information Systems*, 8(3):42–73, 2012.

[28] H. Poon and P. Domingos. Joint inference in information extraction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 913–918, 2007.

[29] K. Raghunathan, H. Lee, S. Rangarajan, N. Chambers, M. Surdeanu, D. Jurafsky, and C. D. Manning. A multi-pass sieve for coreference resolution. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 492–501, 2010.

[30] C. Ré, A. A. Sadeghian, Z. Shan, J. Shin, F. Wang, S. Wu, and C. Zhang. Feature engineering for knowledge base

construction. *IEEE Data Engineering Bulletin*, 37(3):26–40, 2014.

[31] S. Riedel, L. Yao, and A. McCallum. Modeling relations and their mentions without labeled text. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD)*, pages 148–163, 2010.

[32] E. Riloff. Automatically generating extraction patterns from untagged text. In *AAAI/IAAI, Vol. 2*, pages 1044–1049, 1996.

[33] D. Roth and W.-T. Yih. A Linear Programming Formulation for Global Inference in Natural Language Tasks. In *Proceedings of the 2004 Conference on Computational Natural Language Learning (CoNLL)*, pages 1–8, 2004.

[34] E. Sandhaus. *The New York Times Annotated Corpus*. Linguistic Data Consortium, 2008.

[35] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A large ontology from wikipedia and wordnet. *Elsevier Journal of Web Semantics*, 6(3):203–217, 2008.

[36] A. Sun, R. Grishman, W. Xu, and B. Min. New york university 2011 system for kbp slot filling. In *Proceedings of the Text Analysis Conference (TAC)*, 2011.

[37] M. Surdeanu, J. Tibshirani, R. Nallapati, and C. Manning. Multi-instance multi-label learning for relation extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2012.

[38] C. A. Thompson, M. E. Califf, and R. J. Mooney. Active learning for natural language parsing and information extraction. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 406–414, 1999.

[39] J. D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume I*. Computer Science Press, 1988.

Appendix

$$\mathsf{r}(\mathsf{t}) \Leftarrow \mathsf{str2span}(\text{`Lee Harvey Oswald'}, \mathsf{s}) \wedge \mathsf{span2pos}(\mathsf{s}, \mathsf{p}) \wedge \mathsf{nsubj}(\mathsf{c}, \mathsf{p}) \wedge \mathsf{token}(\mathsf{c}, \mathsf{t})$$

$$\downarrow \text{translation}$$

```
SELECT ti4.tokenID
FROM tokenInst ti0, tokenInst ti1,
  tokenInst ti2, tokenInst ti3,
  dependencyInst di0, tokenInst ti4
WHERE ti4.offset = di0.from
  AND ti4.sentenceID = di0.sentenceID
  AND di0.to = ti3.offset
  AND di0.sentenceID = ti3.sentenceID
  AND di0.dependencyID = 11
  AND ti3.offset < ti2.offset + 1
  AND ti3.offset >= ti0.offset
  AND ti3.sentenceID = ti0.sentenceID
  AND ti2.tokenID = 79216
  AND ti1.tokenID = 6058
  AND ti0.tokenID = 5322
  AND ti0.offset + 2 = ti2.offset
  AND ti0.sentenceID = ti2.sentenceID
  AND ti0.offset + 1 = ti1.offset
  AND ti0.sentenceID = ti1.sentenceID
```

Figure 5: Translation of a (safe) expression in first-order logic to SQL. The expression returns verbs for which Lee Harvey Oswald appears as subject. `str2spans` and `span2pos` are intensional predicates, `nsubj` and `token` are extensional. Each predicate gets translated into a fragment of SQL; the fragments are then combined into a single SQL query, which can be efficiently executed.

```
killOfNom(c, b) ⇐ . . .
. . . prep-of(c, b) ∧ token(c, 'assassination')
. . . prep-of(c, b) ∧ token(c, 'execution')
. . . prep-of(c, b) ∧ token(c, 'felling')
. . . prep-of(c, b) ∧ token(c, 'killing')
. . . prep-of(c, b) ∧ token(c, 'shooting')
. . . prep-of(c, b) ∧ token(c, 'slaughter')
. . . prep-of(c, b) ∧ token(c, 'slaying')
. . . prep-of(c, b) ∧ token(c, 'stabbing')
. . . prep-of(c, b) ∧ token(c, 'murder')
. . . nn(c, b) ∧ token(c, 'assassination')
. . . nn(c, b) ∧ token(c, 'murder')
. . .
killerRole(c) ⇐ . . .
. . . token(c, 'assassin')
. . . token(c, 'murderer')
. . .
killingBNF(c, b) ⇐ . . .
. . . dobj(c, b) ∧ token(c, 'assassinate')
. . . dobj(c, b) ∧ token(c, 'murder')
. . .
killingBInf(d, b) ⇐ . . .
. . . dobj(d, b) ∧ token(d, 'assassinating')
. . . dobj(d, b) ∧ token(d, 'murdering')
. . .
killed(a, b) ⇐ person(b) ∧ person(a) ∧ (a ≠ b) ∧ . . .
. . . actInd(a, c, 'confess') ∧ prepc-to(c, d) ∧ killingBInf(d, b)
. . . actInd(a, c, 'confess') ∧ prep-to(c, d) ∧ killOfNom(d, b)
. . . actInd(a, c, 'assassinate') ∧ dobj(c, b)
. . . actInd(a, c, 'murder') ∧ dobj(c, b)
. . .
. . . agent(c, a) ∧ partmod(b, c) ∧ token(c, 'assassinated')
. . . agent(c, a) ∧ partmod(b, c) ∧ token(c, 'murdered')
. . .
. . . agent(c, a) ∧ rcmod(b, c) ∧ token(c, 'assassinated')
. . . agent(c, a) ∧ rcmod(b, c) ∧ token(c, 'murdered')
. . .
. . . appos(a, c) ∧ poss(c, b) ∧ killerRole(c)
. . . appos(a, c) ∧ prep-of(c, b) ∧ killerRole(c)
. . . appos(c, a) ∧ poss(c, b) ∧ killerRole(c)
. . . appos(c, a) ∧ prep-of(c, b) ∧ killerRole(c)
. . . dep(a, c) ∧ dobj(c, b) ∧ token(c, 'assassinated')
. . . dep(a, c) ∧ dobj(c, b) ∧ token(c, 'murdered')
. . . infmod(a, c) ∧ killingBNF(c, b)
. . . nsubj(c, a) ∧ prep-in(c, d) ∧ token(c, 'suspect') ∧ killOfNom(d, b)
. . . nsubj(c, a) ∧ xcomp(c, d) ∧ killingBInf(d, b)
. . . partmod(a, c) ∧ killingBInf(c, b)
. . . partmod(a, c) ∧ prepc-for(c, d) ∧ token(c, 'sentenced') ∧ killingBInf(d, b)
. . . partmod(a, c) ∧ prepc-of(c, d) ∧ token(c, 'accused') ∧ killingBInf(d, b)
. . . partmod(a, c) ∧ prepc-of(c, d) ∧ token(c, 'convicted') ∧ killingBInf(d, b)
. . . partmod(a, c) ∧ prepc-with(c, d) ∧ token(c, 'charged') ∧ killingBInf(d, b)
. . . partmod(a, c) ∧ prep-in(c, d) ∧ token(c, 'wanted') ∧ prep-with(d, e) ∧ token(d, 'connection') ∧
    killOfNom(e, b)
. . . partmod(a, c) ∧ prep-to(c, d) ∧ token(c, 'linked') ∧ killOfNom(d, b)
. . . passInd(a, c, 'accuse') ∧ prep-of(c, d) ∧ killOfNom(d, b)
. . . passInd(a, c, 'charge') ∧ prepc-with(c, d) ∧ killingBInf(d, b)
. . . passInd(a, c, 'charge') ∧ prep-with(c, d) ∧ killOfNom(d, b)
. . . passInd(a, c, 'convict') ∧ prepc-for(c, d) ∧ killingBInf(d, b)
. . . passInd(a, c, 'convict') ∧ prepc-of(c, d) ∧ killingBInf(d, b)
. . . passInd(a, c, 'convict') ∧ prepc-of(c, d) ∧ prep-in(d, e) ∧ token(d, 'taking') ∧ killOfNom(e, b)
. . . passInd(a, c, 'convict') ∧ prep-for(c, d) ∧ killOfNom(d, b)
. . . passInd(a, c, 'convict') ∧ prep-in(c, d) ∧ prep-of(d, b) ∧ token(d, 'death')
. . . passInd(a, c, 'convict') ∧ prep-of(c, d) ∧ killOfNom(d, b)
. . . passInd(a, c, 'link') ∧ prep-to(c, d) ∧ killOfNom(d, b)
. . . passInd(a, c, 'sentence') ∧ prep-for(c, d) ∧ killOfNom(d, b)
. . . passInd(a, c, 'want') ∧ prep-in(c, d) ∧ prep-with(d, e) ∧ token(d, 'connection') ∧ killOfNom(e, b)
. . . passInd(b, c, 'assassinate') ∧ agent(c, a)
. . . passInd(b, c, 'gun') ∧ prt(c, d) ∧ token(d, 'down') ∧ agent(c, a)
. . . passInd(b, c, 'murder') ∧ agent(c, a)
. . . passInd(l, c, 'take') ∧ token(l, 'life') ∧ poss(l, b) ∧ agent(c, a)
. . . passInd(l, c, 'take') ∧ token(l, 'life') ∧ prep-of(l, b) ∧ agent(c, a)
. . .
. . . poss(c, a) ∧ killOfNom(c, b)
. . . poss(c, a) ∧ nsubjpass(d, c) ∧ token(c, 'name') ∧ prep-to(d, e) ∧ token(d, 'linked') ∧ killOfNom(e, b)
. . . prep-by(c, a) ∧ killOfNom(c, b)
. . . rcmod(a, c) ∧ dobj(c, b) ∧ token(c, 'assassinated')
. . . rcmod(a, c) ∧ dobj(c, b) ∧ token(c, 'murdered')
. . .
. . . token(a, 'suspect') ∧ nsubj(c, a) ∧ prep-in(c, d) ∧ prep-of(d, b) ∧ nn(e, d) ∧ token(e, 'murder') ∧
    token(d, 'trial')
. . . xsubj(c, a) ∧ killingBNF(c, b)
. . .
. . .
```

Figure 6: Selected extraction rules created for relation $killed$ during the experiment. Many extractions were obtained during the *Bootstrap* phase, which suggested rules combining syntactic dependencies (eg. nn) and lexical information (eg. token). Users selected from these suggestions, but also adapted them by adding constraints (eg. prt, $\neq$, person). *WordSim* added lexical variety (eg. killOfNom), and *Linguistics* covered additional verb inflections (encoded by predicates actInd and passInd). *Composition* introduced re-usable components (eg. killerRole, killingBNF).